SHORT COURSE ON RELATIONAL DATABASES

Brian W. Bush
TSA-4, Energy and Environmental Analysis
Los Alamos National Laboratory

October 3, 1994

PURPOSE

- To provide a detailed introduction to some advanced aspects of relational databases such as
  - relational algebra
  - design methodology
- To show what approaches, tools, and methodologies exist in this field and what use they are.
- To serve the needs of
  - persons wanting an introduction to relational databases.
  - relational database users lacking a formal background for them.
- This course is not about specific products, hardware, or implementation issues.

SCHEDULE

- Six one-hour meetings, but may be more or fewer depending on how fast we cover the material and what the interests of those taking the course are.
- Informal.
- Readings and exercises will be provided.
- Work outside the course will be necessary to master the material.

OUTLINE

I.	Database concepts	Oct. 3
II.	The relational model	
III.	Query languages (QBE and SQL)	Oct. 7
IV.	Entity-relationship modeling	Oct. 17
V.	Functional dependency	Oct. 24
VI.	Normalization	
VII.	Distributed databases	Oct. 31
VIII.	Database APIs	Nov. 7
IX.	Object-oriented databases	

I. DATABASE CONCEPTS

DEFINITIONS*

- *Database:* A database is an organized collection of data. It contains the data necessary for some purpose (or collection of purposes). The data is arranged to facilitate some set of activities. The data is arranged so that it may be accessed and altered in an efficient manner.

- *Database Management System (DBMS):* A database management system is software that manages data.

* After *C. J. Wertz*.

WHY USE A DATABASE

- A properly designed database can . . .
  - eliminate redundancy in storage.
  - enforce the consistency of the data.
  - ease the integration and sharing of data.
  - provide data independence.
  - enhance the logical organization of data.
- We generally want these features when dealing with data, so we should use a database instead of reinventing one.

DATABASE CAPABILITIES*

- *Persistence:* The ability of objects to *persist* through different program invocations. Data manipulated by a database can be either *transient* or *persistent*. Transient data are only valid inside a program or transaction; they are lost once the program or transaction terminates. Persistent data are stored outside of a transaction and survive updates. In other words, they persist across transactions, system crashes, and even media crashes (e.g., magnetic disk head crashes). These data are the *recoverable* objects of the database.

* After *S. Khoshafian*.

- *Transactions:* Units executed either entirely or not at all. Transactions are *atomic*. If the user performs updates to the persistent database within a transaction, either *all* of the updates must be visible to the outside world, or *none* of the updates must be seen. In the former case, the transaction has been *committed*. In the latter case, the transaction was *aborted*. Transactions in databases take the DBMS from one consistent state to another.

- *Concurrency control:* In the typical execution environment of a database management system, transactions run concurrently. To guarantee database and transaction consistency, database management systems impose a *serializable* order of execution. That is, the results of transactions are the same as if the transactions were executed one after another (in a series) instead of at the same time. To guarantee serializability of transactions, database management systems use concurrency control strategies.

- *Recovery:* The database management system must guarantee that *partial* updates of transactions that fail are not propagated to the persistent database. There are three types of failures from which a system must recover: transaction, system, or media errors. Reliability and the graceful recovery from these types of failures are important features of a database management system.

- *Querying:* Queries are used to select subsets and subobjects from database collection objects or sets. Queries are expressed in terms of high-level declarative constructs that allow users to qualify *what* they want to retrieve from the persistent databases. Some query languages have solid theoretical foundations (e.g., relational calculus). Other database management systems use more ad hoc query facilities.

- *Versioning:* In database management systems the *same* persistent object undergoes many changes, or state transitions, and it is desirable to access or investigate previous states of the object. Versioning consists of tools and constructs to automate or simplify the construction and organization of versions or configurations.

- *Security:* Database management systems incorporate security primitives for accessing or updating persistent objects. A security mechanism (for example, through a *grant* and *revoke* or privileges) and the protection of persistent databases from adverse access are integral parts of any database management system.

- *Performance issues:* Database management systems use a number of strategies to enhance the overall performance of the system: indices and accelerators, storage management techniques, query optimization, and caching of frequently accessed objects.

TYPICAL DATABASE FEATURES

- data definition language
- data dictionary
- data manipulation language
- query language
- report generator
- integrity
- recovery

DATA MODELS

- A data model is a descriptive representation of a data structure. Data models are sometimes classified according to purpose. They are sometimes classified according to modeling style. A complete classification requires a specification of purpose and style.*

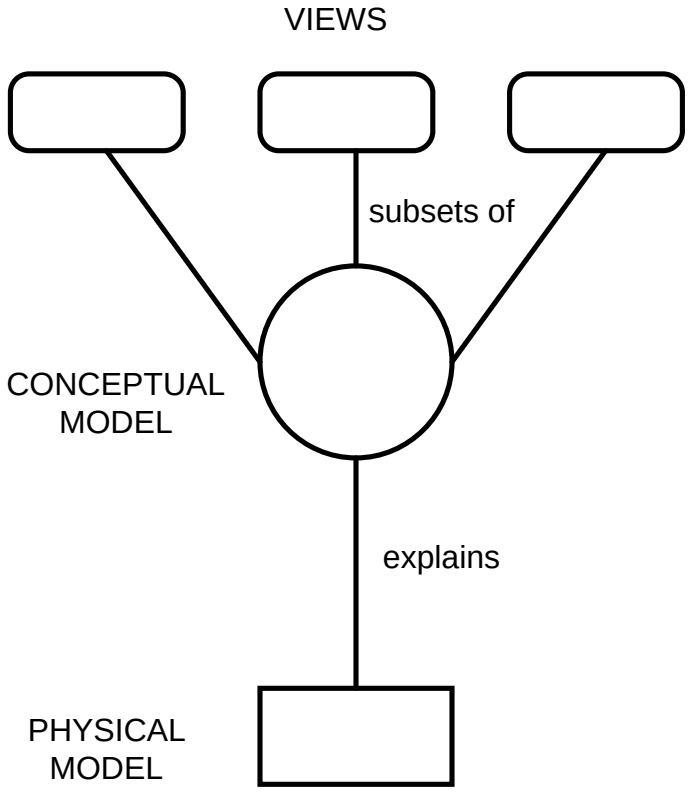
- Data Model ≡ Database Schema.

- Types of models:
  - physical / internal
  - conceptual / logical
  - external / [view]

* After *C. J. Wertz*.

ANSI/SPARC ARCHITECTURE*

- ANSI Standards Planning and Requirements Committee of X3 (Data Management) proposed a three schema architecture in 1970:
  - A physical model is a representation of the database as it is stored in machine readable form.
  - A conceptual model illustrates the data content of the database in a form suitable for interpretation by humans.
  - A view is a representation of a subset of the data organized for a specific user or purpose.



* After *C. J. Wertz*.

ANALYSIS / DESIGN*

- Database and computer system design involve identifying and defining individual data elements and devising logical groupings for storing and processing them.

- *Bottom up approach:* identifying and defining all atoms of data, utilizing user views as a means for organizing the elements and synthesizing or integrating them into a database design.

- *Top down approach:* identifying the things or entities of interest, the relationship between the entities and the data elements which describe each. This conceptual model then becomes the basis for the database design.

* After *C. J. Wertz*.

- *Combination approach:*
  - 0. Requirements analysis.
  - 1. Development of entity model.
  - 1. Identify the required user views along with all elements in each view.
  - 2. Development of normalized model.
  - 3. Augmenting the logical model with data volumes and accessing patterns.

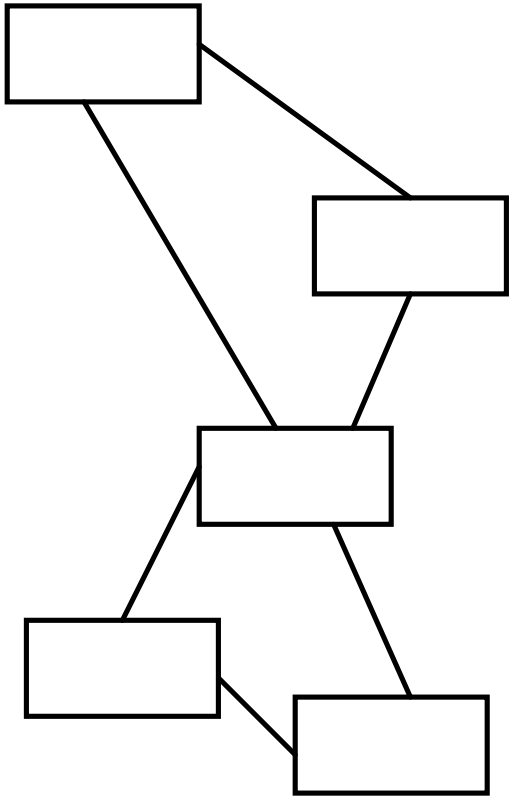
FILE MANAGEMENT SYSTEM DATABASES

- Early databases focused on file management routines (sorting, maintenance, report generation, file description) and were closely tied to the file management system.

- History:
  - 1950/60s data definition products
  - 1950/60s IBM, GE, Honeywell products
  - 1950/60s COBOL developed by CODASYL

NETWORK DATABASES

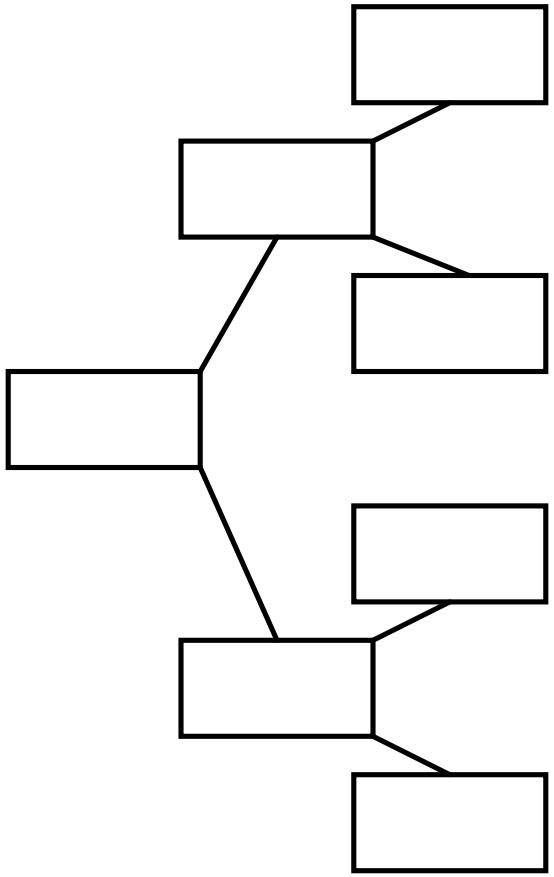
- Different record types available.
- One-to-many relationships among record types.
- History:
  - 1960s GE IDS (Integrated Data Store)
  - 1969 DBTG (Data Base Task Group of CODASYL) develops DML (Data Manipulation Language)—the foundation for network DBMS
  - 1970 IDMS from Cullinet
  - 1971 DMS 1100 from Sperry
  - 1975 IDS-2 Honeywell



- Still in use today because of their flexibility, standardization, and performance.

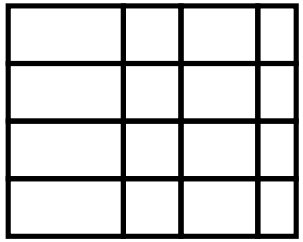
HIERARCHICAL DATABASES

- Tree-like organization.
- Each record type participates in only one kind of relationship.
- History:
  - 1960s IMS (Information Management System) from IBM
  - 1960s TSDMS (Time-Shared Database Management System) by SDC
- Still in use today because of their high performance and in spite of their inflexibility. (IMS is currently used on 25% of IBM mainframes.)



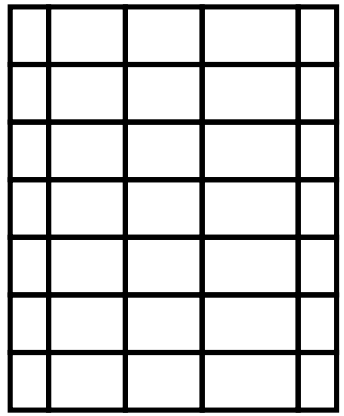
RELATIONAL DATABASES

- High-level declarative style independent of implementation



- Mathematically based on predicate calculus.

- History:
  - 1969 E. F. Codd papers start
  - 1974 System/R by IBM
  - 1974 SEQUEL
  - 1979 ORACLE
  - 1980s SQL/DS by IBM
  - 1981 INGRES
  - 1983 DB2 by IBM
  - 1986 ANSI SQL standard (based on DB2 dialect)

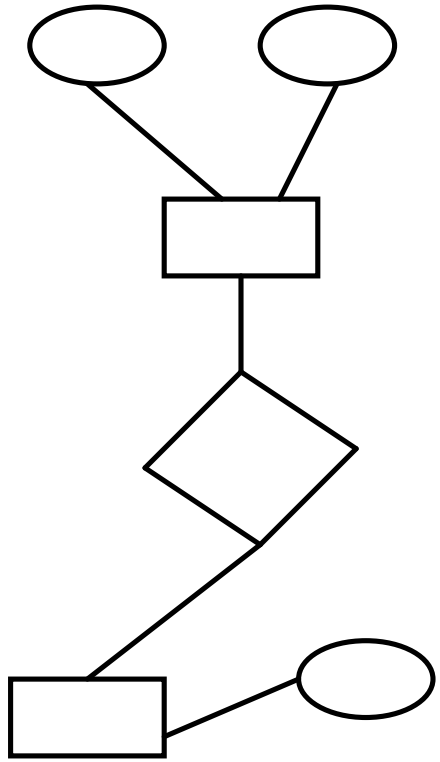


  - 1987 ISO SQL standard
  - 1988 SQL for OS/2 form Ashton-Tate/Microsoft

- The dominant database technology.

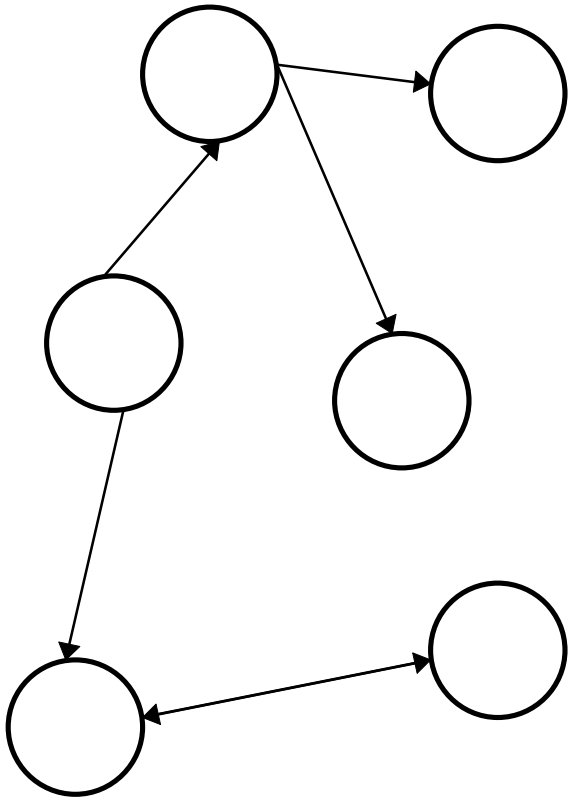
SEMANTIC DATABASES

- Attempt to model real world as closely as possible.
- Based on things of interest, their relationships, and their data.
- History:
  - 1976 Entity-relationship modeling by Chen
  - 1981 DAPLEX
- Few commercial products available.



COMPLEX OBJECT DATABASES

- Object-oriented approach.
- History:
  - 1983 VERSO
  - 1983 GEM
  - 1984 LDM
  - 1986 GemStone by Sevio
  - 1987 Gbase by Grapahel
  - 1987 Vbase
- Growing commercial importance.



II. THE RELATIONAL MODEL

RELATIONAL MODEL

- The relational data model was designed to enhance data independence and consistency by severing the connection to physical storage devices and by providing a rigorous definition of both data structure and data manipulation.*

- The relational model is *a way of looking at data*; it can be regarded as a prescription for how data might be represented and how that representation might be manipulated. More specifically, the model is concerned with three aspects of data: data *structure*, data *integrity*, and data *manipulation*.†

* After *C. J. Wertz*.
† After *C. J. Date*.

BASIC TERMS

- The relational model views data in a tabular format:
  - table ≡ relation
  - column ≡ attribute
  - row ≡ tuple
  - heading of table ≡ names of attributes
  - content ≡ data
  - database ≡ set of named tables/relations
- The heading of the table is its relational schema.
- The set of all relational schemas is the database schema.
- The ordering of rows or columns does not convey information.

- Example:

T		
A	*B*	*C*
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4

  - **T** is a relation (table)
  - Head(**T**) = (A, B, C) is the relational schema (heading)
  - A, B, and C are attributes (columns)
  - (a2, b2, c2) is a tuple of the relation (row of the table), part of the content

DOMAINS AND DATATYPES

- Each attribute has a definite type.
- The of domain of an attribute is the set of values it can have.
- Domains express a constraint.
- Examples:
  - colors
  - cities
  - money
  - even integers
  - part number

TABLES AND RELATIONS

- A relation is the subset of a Cartesian product of domains.
- Example: $C \subseteq A \times B$

C	
A	*B*
a1	b1
a2	b2

$\subseteq$

A	*B*
a1	b1
a1	b2
a2	b1
a2	b2

=

A
a1
a2

$\times$

B
b1
b2

- Since not all component values sit with all other component values on a tuple of a relation, those that do are said to be *related*.*

* After *P. O'Neil*.

KEYS AND SUPERKEYS

- *Superkey:* any two rows will always have distinct values for this subset of columns.
- *Key:* no proper subset of the columns is a superkey.
- *Candidate key:* any key in a relation.
- *Primary key:* a candidate key chosen to uniquely specify table rows.
- *Foreign key:* An attribute that occurs as the primary key of another relation.
- Keys express a constraint. The contents of a table do not define what the keys are.

- Example:

Hardware			
Property Number	*Serial Number*	*Description*	*Location*
901551	439A124	Gateway 2000	TA-3 SM-43 D14
902311	4551GXB	Aquiline 386	TA-3 SM-43 A3D

Places		
Location	*Group*	*Owner*
TA-3 SM-43 D14	TSA-4	Brian Bush
TA-3 SM-43 A3D	TSA-4	Fred Roach

  - (Property Number, Serial Number, User) is a superkey (there are many others) for **Hardware**
  - (Property Number) and (Serial Number) are candidate keys for **Hardware**
  - (Property Number) is the primary key for **Hardware**
  - (Location) is a foreign key in **Hardware** (assuming that (Location) is the primary key in **Places**)

NULL VALUES

- A *null value* is a special value that has the interpretation of “unknown,” “not yet defined,” or “inapplicable.”
- Null values affect averages and counting.
- The primary key cannot have a null value.
- Example:

Hardware			
Property Number	*Serial Number*	*Description*	*Location*
901551	NULL	Gateway 2000	TA-3 SM-43 D14
902311	4551GXB	Aquiline 386	TA-3 SM-43 A3D

  - Count(Property Number) = 2, but Count(Serial Number) = 1

RELATIONAL ALGEBRA

- E. F. Codd, 1970.

- Set theoretic operations:

Name	*Symbol*	*Reading*	*Example*
UNION	$\cup$	UNION	R $\cup$ S
INTERSECTION	$\cap$	INTERSECT	R $\cap$ S
DIFFERENCE	–	MINUS	R – S
PRODUCT	$\times$	TIMES	R $\times$ S

- Relational operations:

Name	*Symbol*	*Reading*	*Example*
PROJECT	[ ]	[ ]	R[A_5, . . ., A_9]
SELECT	where	where	R where A_1 = 5
JOIN	$\triangleright\triangleleft$	JOIN	R $\triangleright\triangleleft$ S
DIVISION	$\div$	DIVIDE BY	R $\div$ S

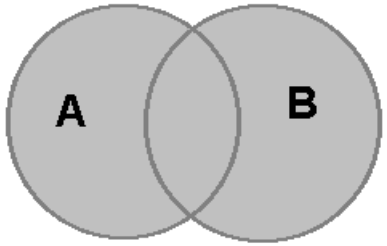
- Assignment: ":="

- Precedence:

Precedence	*Operators*	*Symbols*
Highest	project	[ ]
	select	where
	times	$\times$
	join, divide by	$\triangleright\triangleleft$, $\div$
	difference	–
Lowest	union, intersection	$\cup$, $\cap$

SET THEORETIC OPERATIONS*

- Union: The union of two compatible tables **A** and **B** is the table **A** ∪ **B**, with the same heading, consisting of all rows that are in **A** or **B** or both.



A	
X	*Y*
x1	y1
x2	y2
x3	y3

∪

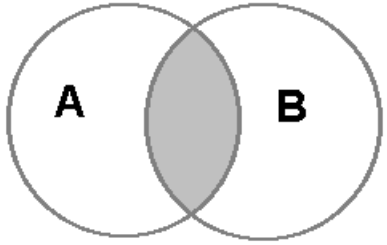
B	
X	*Y*
x1	y1
x3	y3
x5	x5

=

A ∪ B	
X	*Y*
x1	y1
x2	y2
x3	y3
x5	y5

* After *P. O'Neil*.

- Intersection: The intersection of two compatible tables **A** and **B** is the table **A** ∩ **B**, with the same heading, consisting of all rows that are in both **A** and **B**.



A	
X	*Y*
x1	y1
x2	y2
x3	y3

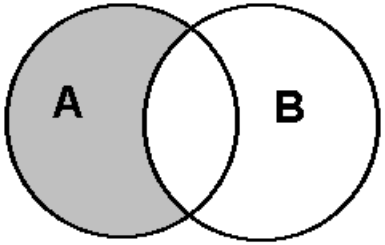
∩

B	
X	*Y*
x1	y1
x3	y3
x5	x5

=

A ∩ B	
X	*Y*
x1	y1
x3	y3

- Difference: The difference between two compatible tables **A** and **B** is the table **A** – **B**, with the same heading, consisting of all rows that appear in **A** but do not appear in **B**.



A	
X	*Y*
x1	y1
x2	y2
x3	y3

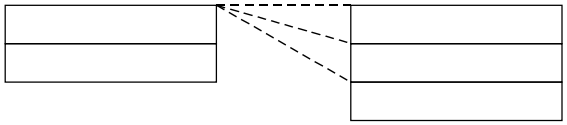
–

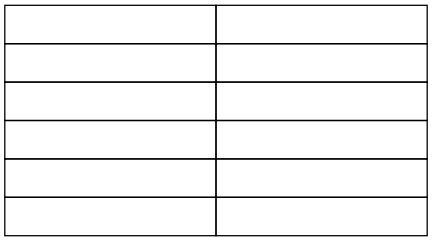
B	
X	*Y*
x1	y1
x3	y3
x5	x5

=

A – B	
X	*Y*
x2	y2

- Product: The product of tables **A** and **B** is the table **A** $\times$ **B** whose heading is (A_1, . . ., A_n, B_1, . . ., B_n) and whose content is all rows (a_1, . . ., a_n, b_1, . . ., b_n) where (a_1, . . . a_n) and (b_1, . . ., b_n) are any rows of **A** and **B**, respectively.





A	
X	*Y*
x1	y1
x2	y2

$\times$

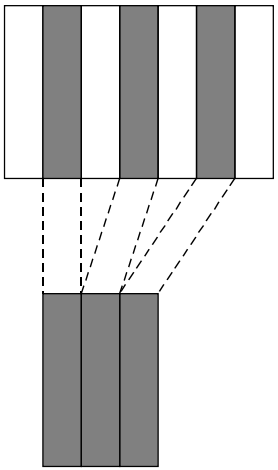
B	
X	*Y*
x1	y1
x3	y3
x5	x5

=

A $\times$ **B**			
A.X	*A.Y*	*B.X*	*B.Y*
x1	y1	x1	y1
x1	y1	x3	y3
x1	y1	x5	y5
x2	y2	x1	y1
x2	y2	x3	y3
x2	y2	x5	y5

RELATIONAL OPERATIONS

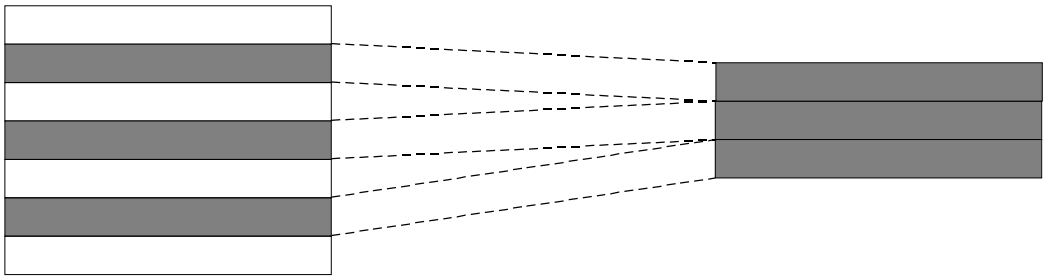
- Projection: The projection of table **A** on attributes A_i, A_j, . . . is the table **A[A_i, A_j, . . .]** whose heading is (A_i, A_j, . . .) and whose content is the rows (a_i, a_j, . . .) for which (a_1, a_2, . . .) are rows in **A**.



A		
X	*Y*	*Z*
x1	y1	z1
x2	y2	z2
x3	y3	z3

A[X, Y]	
X	*Y*
x1	y1
x2	y2
x3	y3

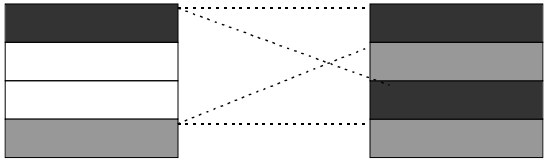
- Selection: The selection of the table **A** by the condition *C* is the table "**A** where *C*" with the same heading as **A** and containing only the rows of **A** for which the condition *C* is true.

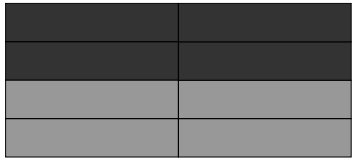


A		
X	*Y*	*Z*
x1	y1	z1
x2	y2	z2
x3	y3	z3

A where X = x1 or Y = y2		
X	*Y*	*Z*
x1	y1	z1
x2	y2	z2

- Join: The join of tables **A** and **B** with headings Head(**A**) = (A_1, . . ., A_n, C_1, . . ., C_k) and Head(**B**) = (B_1, . . . B_m, C_1, . . ., C_k) is the table A ▷◁ B with heading (A_1, . . ., A_n, B_1, . . . B_m, C_1, . . ., C_k) and containing a row (a_1, . . ., a_n, b_1, . . . b_m, c_1, . . ., c_k) if and only if **A** contains the row (a_1, . . ., a_n, c_1, . . ., c_k) and **B** contains the row (b_1, . . . b_m, c_1, . . ., c_k).





A		
X	*Y*	*Z*
x1	y1	z1
x2	y2	z2
x3	y3	z3
x4	y4	z4
x5	y6	z7

▷◁

B		
X	*Y*	*W*
x1	y1	w1
x3	y3	w3
x5	y5	z5

=

A ▷◁ B			
X	*Y*	*Z*	*W*
x1	y1	z1	w1
x3	y3	z3	w3

- Outer Join: Same as join, except unmatched rows also appear in result.

A		
X	*Y*	*Z*
x1	y1	z1
x2	y2	z2
x3	y3	z3
x4	y4	z4
x5	y6	z7

$\triangleright\triangleleft_O$

B		
X	*Y*	*W*
x1	y1	w1
x3	y3	w3
x5	y5	w5

=

A $\triangleright\triangleleft_O$ B			
X	*Y*	*Z*	*W*
x1	y1	z1	w1
x3	y3	z3	w3
x2	y2	z2	NULL
x4	y4	z4	NULL
x5	y6	z7	NULL
x5	y5	NULL	w5

- Left Outer Join: Same as join, except unmatched rows of the left operand also appear.

A		
X	*Y*	*Z*
x1	y1	z1
x2	y2	z2
x3	y3	z3
x4	y4	z4
x5	y6	z7

$\triangleright\triangleleft_{LO}$

B		
X	*Y*	*W*
x1	y1	w1
x3	y3	w3
x5	y5	w5

=

A $\triangleright\triangleleft_{LO}$ B			
X	*Y*	*Z*	*W*
x1	y1	z1	w1
x3	y3	z3	w3
x2	y2	z2	NULL
x4	y4	z4	NULL
x5	y6	z7	NULL

- Right Outer Join: Same as join, except unmatched rows of the right operand also appear.

A		
X	*Y*	*Z*
x1	y1	z1
x2	y2	z2
x3	y3	z3
x4	y4	z4
x5	y6	z7

$\triangleright\triangleleft_{RO}$

B		
X	*Y*	*W*
x1	y1	w1
x3	y3	w3
x5	y5	w5

=

A $\triangleright\triangleleft_{RO}$ B			
X	*Y*	*Z*	*W*
x1	y1	z1	w1
x3	y3	z3	w3
x5	y5	NULL	w5

- Theta Join: For $\theta \in \{=, <, >, \ldots\}$ and **A** and **B** have no column names in common, the theta join is defined as
  $\mathbf{A} \rhd\lhd_{\mathbf{A}_i \ \theta \ \mathbf{B}_j} \mathbf{B} \equiv (\mathbf{A} \times \mathbf{B})$ where $\mathbf{A_i} \ \theta \ \mathbf{B_j}$

- Division: The result of the division of **A** by **B** is the table $\mathbf{A} \div \mathbf{B}$ that contains the largest possible set of rows such that $(\mathbf{A} \div \mathbf{B}) \times \mathbf{B} \subseteq \mathbf{A}$.

A		
X	*Y*	*Z*
x1	y1	z1
x2	y2	z1
x2	y2	z2
x3	y3	z1
x3	y3	z2

÷

B
Z
z1
z2

=

A ÷ B	
X	*Y*
x2	y2
x3	y3

CODD'S TWELVE RULES*

- *1. Information rule:* All information in a relational database is represented explicitly at the logical level in exactly one way—by values in tables.

- *2. Guaranteed access rule:* Each and every datum (atomic value) in a relation database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.

- *3. Systematic treatment of null values:* Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

* After *E. F. Codd*.

- *4. Dynamic on-line catalog based on the relational model:* The database's description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

- *5. Comprehensive data sublanguage rule:* A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings, and which is comprehensive in supporting all of the following items:
  - data definition
  - view definition
  - data manipulation (interactive and by program)
  - integrity constraints
  - authorization
  - transaction boundaries (begin, commit, and rollback)

- *6. View updating rule:* All views that are theoretically updatable are also updatable by the system.

- *7. High-level insert, update, and delete:* The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update , and deletion of data.

- *8. Physical data independence:* Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.

- *9. Logical data independence:* Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

- *10. Integrity independence:* Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

- *11. Distribution independence:* A relational DBMS has distribution independence.

- *12. Nonsubversion rule:* If a relational system has a low-level (single record at a time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher-level relational language (multiple records at a time).

VB to SQL Server Strategies | Multidatabase APIs | JAM 6

DBMS

DATABASE & CLIENT/SERVER SOLUTIONS

OCTOBER 1994
VOLUME 7 NUMBER 11

The Relational Model Turns 25

...And We're Still Trying To Get It Right

INTERVIEW

C.J. Date on The Third Manifesto

NEW AND REVIEWED

Capsule, db-UIM/X, and Access Developer's Toolkit

III. QUERY LANGUAGES (QBE and SQL)

QUERY LANGUAGES

- A database query language is an integral part of a DBMS, enabling users to interact with the system.

- QBE (Query by Example) and SQL (Structured Query Language) both emerged in the mid-1970s in connection with IBM. They were designed specifically for the relational model and are now the most commonly used query languages.

SQL STANDARDS

- There is an ANSI/ISO standard that is being continually revised (SQL-89, SQL-93, SQL3, etc.)
- There are also competing standards from IBM and X/OPEN.
- The standards have some fairly large “holes” in them:
  - error codes
  - data types
  - system tables
  - interactive SQL
  - programmatic interface
  - dynamic SQL
  - semantic differences
  - collating sequences
  - database structure

LEADING SQL-BASED DBMSs

Name	*Platforms*
DB2	IBM mainframes under MVS
SQL/DS	IBM mainframes under VM and DOS/VSE
Rdb/VMS	Digital VAX/VMS minicomputers
Oracle	Mainframe, mincomputers, Pcs
Sybase	Mincomputers and PC LANs
Informix-SQL	UNIX-based mincomputers and PCs
Unify	UNIX-based minicomputers
OS/2 Extended Edition	IBM PS/2 systems under OS/2
SQL Server	OS/2-based PC LANs
SQLBase	DOS- and OS/2-based PC LANs
dBASE IV	PCs and PC LANs
Paradox	PCs and PC LANs

SQL SELECT SYNTAX*

- Subselect general form:

  select [all|distinct] *expression* {, *expression*}
  from *tablename* [*corr_name*] {, *tablename* [*corr_nam*]}
  [where *search_condition*]
  [group by *column* {, *column*}]
  [having *search_condition*]

- Full select general form:

  subselect
  {union [all] *subselect*}
  [order by *result_column* [asc|desc] {, *result_column* [asc|desc]}]

* After *P. O'Neil*.

CONCEPTUAL ORDER OF EVALUATION OF SQL SELECT*

- First the Cartesian product of all tables in the *from* clause is formed.
- From this, rows not satisfying the *where* condition are eliminated.
- The remaining rows are grouped in accordance with the *group by* clause.
- Groups not satisfying the *having* clause are then eliminated.
- The expressions of the *select* clause target list are evaluated.
- If the key word *distinct* is present, duplicate rows are now eliminated.
- The *union* is taken after each subselect is evaluated.

* After *P. O'Neil*.

- Finally, the set of all selected rows is sorted if an *order by* is present.

SQL FUNCTIONS*

- set functions

Name	*Argument Type*	*Result Type*	*Description*
count	any (can be *)	numeric	count of occurrences
sum	numeric	numeric	sum of arguments
avg	numeric	numeric	average of arguments
max	character or numeric	same as argument	maximum value
min	character or numeric	same as argument	minimum value

- arithmetic functions
- character functions

* After *P. O'Neil*.

SQL PREDICATES*

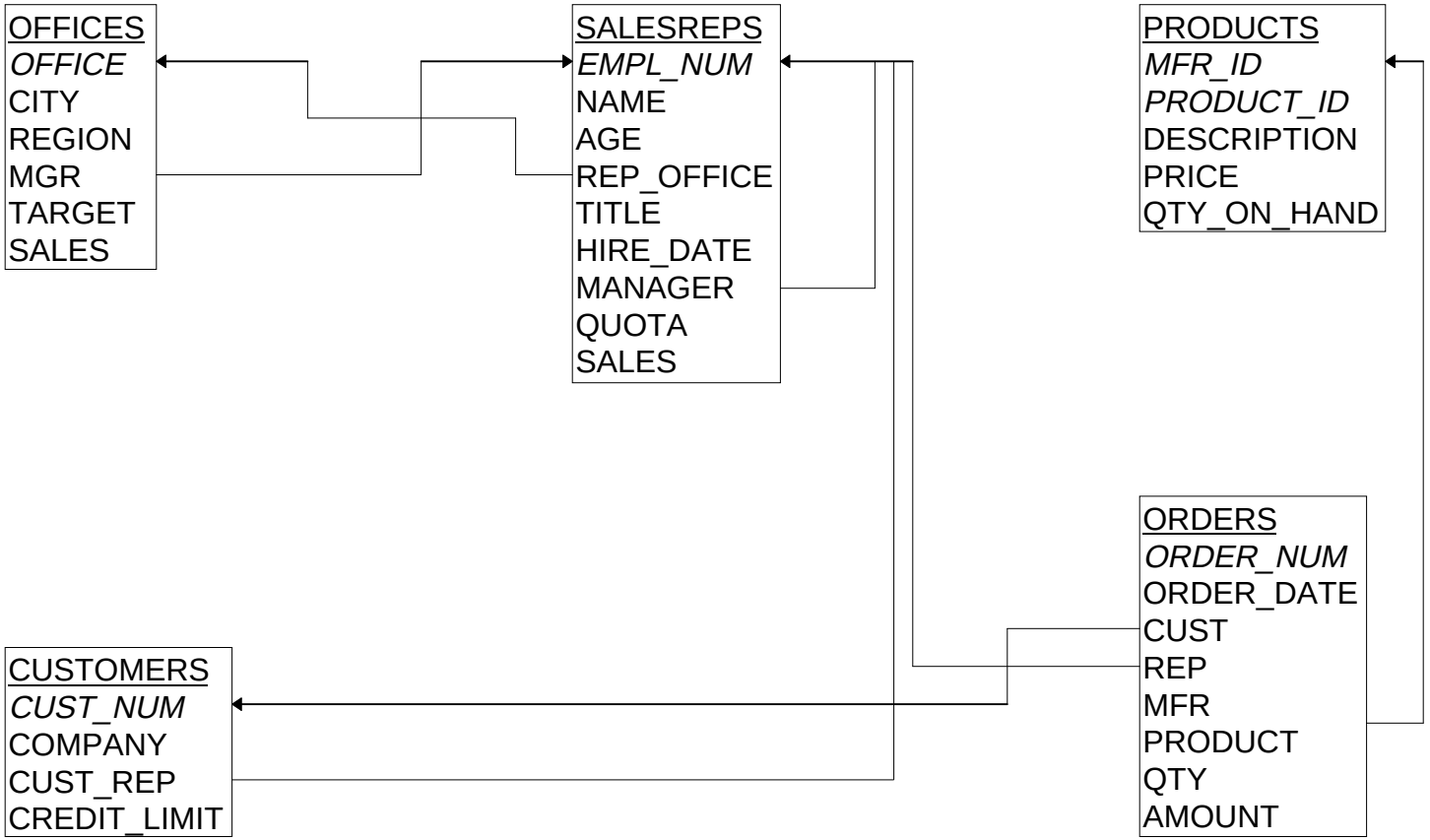
- Comparison operators: $\theta \in$ {=, <>, >, >=, < <=}.
- Predicates:

Predicate	*Form*	*Example*
comparison	*expr1* θ (*expr2*|*subselect*)	p.price > (*subselect*)
between	*expr1* [not] between *expr2* and *expr3*	c.discnt between 10.0 and 12.0
quantified	*expr* θ[all|any] (*subselect*)	c.discnt >=all (*subselect*)
in	*expr* [not] in (*subselect*)	pid in (selct pid from orders)
exists	[not] exists (*subselect*)	exists (select * . . .)
is null	*columname* is [not] null	c.discnt is null
like	*columnname* [not] like '*pattern*'	cname like 'A%'

- Truth values: true, false, and unknown.

* After *P. O'Neil*.

SAMPLE DATABASE*



* After *J. R. Groff* and *P. N. Weinberg*.

EXAMPLE SQL QUERIES*

- Simple selection: *List the sales offices with their targets and actual sales.*

```
Select CITY, TARGET, SALES
    From OFFICES
```

- Simple search condition: *List the Eastern region sales offices with their targets and sales.*

```
Select CITY, TARGET, SALES
    From OFFICES
    Where REGION = 'Eastern'
```

* After *J. R. Groff* and *P. N. Weinberg*.

- Sorting of result: *List Eastern region sales offices whose sales exceed their targets, sorted in alphabetical order.*

```
Select CITY, TARGET, SALES
    From OFFICES
    Where REGION = 'Eastern' And SALES > TARGET
    Order By CITY
```

- Set functions: *What are the average target and sales for Eastern region offices?*

```
Select avg(TARGET), avg(SALES)
    From OFFICES
    Where REGION = 'Eastern'
```

- Computations: *List the city, region, and amount over/under target for each office.*

```
Select CITY, REGION, (SALES - TARGET)
    From OFFICES
```

- Selecting all columns: *Show all the data in the OFFICES table.*

```
Select *
    From OFFICES
```

- Eliminating duplicates: *List the employee numbers of all sales office managers.*

```
Select MGR
    From OFFICES
```

- Single-row retrieval: *Retrieve the name and credit limit of customer number 2107.*

```
Select COMPANY, CREDIT_LIMIT
    From CUSTOMERS
    Where CUST_NO = 2107
```

- Simple set membership: *List the salespeople who work in New York, Atlanta, or Denver.*

```
Select NAME, QUOTA, SALES
    From SALESREPS
    Where REP_OFFICE In (11, 13, 22)
```

- Pattern matching: *Show the credit limit of companies whose name starts with "Smith."*

```
Select COMPANY, CREDIT_LIMIT
    From CUSTOMERS
    Where COMPANY LIKE "Smith%"
```

- Null value test: *Find the salespersons not yet assigned to an office.*

```
Select NAME
    From SALESREPS
    Where REP_OFFICE Is Null
```

- Combining query results: *List all the products where the price of the product exceeds $2000 or where more than $30,000 of the product has been ordered on a single order.*

```
Select MFR_ID, PRODUCT_ID
        From PRODUCTS
        Where PRICE > 2000.00
Union Select Distinct MFR, PRODUCT
        From ORDERS
        Where AMOUNT > 30000.00
```

- Equi-joins: *List all orders showing order number, amount, customer name, and the customer's credit limit.*

```
Select ORDER_NUM, AMOUNT, COMPANY, CREDIT_LIMIT
        From ORDERS, CUSTOMERS
        Where CUST = CUST_NUM
```

- Non-equi-joins: *List all combinations of salespeople and offices where the salesperson's quota is more than the office's target.*

```
Select NAME, QUOTA, CITY, TARGET
    From SALESREPS, OFFICES
    Where QUOTA > TARGET
```

- Qualified column names: *Show the name, sales, and office for each salesperson.*

```
Select NAME, SALESREPS.SALES, CITY
    From SALESREPS, OFFICES
    Where REP_OFFICE = OFFICE
```

- Self-joins: *List the names of salespeople and their managers.*

```
Select EMPS.NAME, MGRS.NAME
    From SALESREPS EMPS, SALESREPS MGRS
    Where EMPS.MANAGER - MGRS.EMPL_NUM
```

- Group search conditions: *What is the average order size for each salesperson whose orders total more than $30,000?*

```
Select REP, avg(AMOUNT)
        From ORDERS
        Group By REP
        Having sum(AMOUNT) > 30000.00
```

- Subquery comparison test: *List the salespeople whose quotas are equal to or higher than the target of the Atlanta sales office.*

```
Select NAME
        From SALESREPS
        Where QUOTA >= (Select TARGET
                From OFFICES
                Where CITY = “Atlanta”)
```

- Subquery set membership test: *List the salespeople who work in offices that are over target.*

```
Select NAME
    From SALESREPS
    Where REP_OFFICE In (Select OFFICE
        From OFFICES
        Where SALES > TARGET)
```

- Subquery existence test: *List the offices where there is a salesperson whose quota represents more than 55% of the office's target.*

```
Select CITY
    From OFFICES
    Where Exists (Select *
        From SALESREPS
        Where REP_OFFICE = OFFICE And Quota > (.55 * TARGET))
```

- Subquery quantified test: *List the salespeople who have taken an order that represents more than ten percent of their quota.*

```
Select NAME
    From SALESREPS
    Where (.10 * QUOTA) < Any (Select AMOUNT
        From ORDERS
        Where REP = EMPL_NUM)
```

SQL DATABASE UPDATES*

- Insert statement syntax:

  insert into *tablename* [(*column* {, *column*})]
  [values (*expression* {, *expression*})] | [subselect]

- Update statement syntax:

  update *tablename* [*corr_name*]
  set *column* = {*expression*|null} {, *column* = {*expression*|null}}
  [where *search_condition*]

- Delete statement syntax:

  delete from *tablename* [*corr_name*]
  [where *search_condition*]

* After *P. O'Neil*.

EXAMPLE SQL DATABASE UPDATES*

- Single-row insert: *Insert a new customer for Mr. Jacobsen.*

```
Insert Into
    CUSTOMERS (COMPANY, CUST_NUM, CREDIT_LIMIT, CUST_REP)
    VALUES ('InterCorp', 2126, 15000.00, 111)
```

- Multi-row insert: *Copy old orders into the OLDORDERS table.*

```
Insert Into
    OLDORDERS (ORDER_NUM, ORDER_DATE, AMOUNT)
    Select ORDER_NUM, ORDER_DATE, AMOUNT
        From ORDERS
        Where ORDER_DATE < '01-JAN-90'
```

* After *J. R. Groff* and *P. N. Weinberg*.

- Update: *Transfer all salespeople from the Chicago office to the New York office, and lower their quotas by 10%.*

```
Update SALESREPS
    Set REP_OFFICE = 11, QUOTA = .90 * QUOTA
    Where REP_OFFICE = 12
```

- Delete: *Delete all orders placed before November 15, 1989.*

```
Delete From ORDERS
    Where ORDER_DATE < '15-NOV-89'
```

QBE SYNTAX

- A query is performed by constructing a example table indicating what operations are to be performed.
- Projection: check mark ($\surd$) in the appropriate columns.
- Selection: condition(s) in the appropriate columns.
- Join: appropriate columns of tables linked using example elements (a.k.a. column variables)
- Aggregation: *calc* operator with aggregate expression.
- Calculation: *calc* operator with arithmetic expressions and example elements.
- Subselect: *set* operator with example elements and set expression.

EXAMPLE QBE QUERIES

- Simple selection: *List the sales offices with their targets and actual sales.*

OFFICES		
CITY	*TARGET*	*SALES*
√	√	√

- Simple search condition: *List the Eastern region sales offices with their targets and sales.*

OFFICES			
CITY	*TARGET*	*SALES*	*REGION*
√	√	√	‘Eastern’

- Sorting of result: *List Eastern region sales offices whose sales exceed their targets, sorted in alphabetical order.*

OFFICES			
CITY	*TARGET*	*SALES*	*REGION*
√	√ *t*	√ *> t*	'Eastern'

- Set functions: *What are the average target and sales for Eastern region offices?*

OFFICES		
TARGET	*SALES*	*REGION*
calc avg	calc avg	'Eastern'

- Computations: *List the city, region, and amount over/under target for each office.*

OFFICES			
CITY	*TARGET*	*SALES*	*REGION*
√	*t*	*s* calc *s* - *t*	√

- Selecting all columns: *Show all the data in the OFFICES table.*

OFFICES					
OFFICE	*CITY*	*REGION*	*MGR*	*TARGET*	*SALES*
√	√	√	√	√	√

- Eliminating duplicates: *List the employee numbers of all sales office managers.*

OFFICES
MGR
√

- Single-row retrieval: *Retrieve the name and credit limit of customer number 2107.*

CUSTOMERS		
COMPANY	*CREDIT_LIMIT*	*CUST_NO*
√	√	2107

- Simple set membership: *List the salespeople who work in New York, Atlanta, or Denver.*

SALESREPS			
NAME	*QUOTA*	*SALES*	*REP_OFFICE*
√	√	√	11
√	√	√	13
√	√	√	22

- Pattern matching: *Show the credit limit of companies whose name starts with "Smith."*

CUSTOMERS	
COMPANY	*CREDIT_LIMIT*
√ 'Smith'..	√

- Null value test: *Find the salespersons not yet assigned to an office.*

SALESREPS
NAME
√ blank

- Combining query results: *List all the products where the price of the product exceeds $2000 or where more than $30,000 of the product has been ordered on a single order.*

  [not possible with single QBE]

- Equi-joins: *List all orders showing order number, amount, customer name, and the customer's credit limit.*

ORDERS		
ORDER_NUM	*AMOUNT*	*CUST*
√	√	*c*

CUSTOMERS		
CUST_NUM	*COMPANY*	*CREDIT_LIMIT*
c	√	√

- Non-equi-joins: *List all combinations of salespeople and offices where the salesperson's quota is more than the office's target.*

SALESREPS	
NAME	*QUOTA*
√	√ *q*

OFFICES	
CITY	*TARGET*
√	√ *< q*

- Qualified column names: *Show the name, sales, and office for each salesperson.*

SALESREPS		
NAME	*SALES*	*REP_OFFICE*
√	√	*o*

OFFICES	
OFFICE	*CITY*
o	√

- Self-joins: *List the names of salespeople and their managers.*

SALESREPS		
EMPL_NUM	*NAME*	*MANAGER*
	√ as EMPS_NAME	*n*
n	√ as MGRS_NAME	

- Group search conditions: *What is the average order size for each salesperson whose orders total more than $30,000?*

  [not possible with single QBE]

- Subquery comparison test: *List the salespeople whose quotas are equal to or higher than the target of the Atlanta sales office.*

  [not possible with single QBE]

- Subquery set membership test: *List the salespeople who work in offices that are over target.*

SALESREPS	
NAME	*REP_OFFICE*
√	only *o*

OFFICES		
OFFICE	*TARGET*	*SALES*
o	*t*	*> t*

- Subquery existence test: *List the offices where there is a salesperson whose quota represents more than 55% of the office's target.*

OFFICES		
OFFICE	*CITY*	*TARGET*
o	√	*t*

SALESREPS	
REP_OFFICE	*QUOTA*

o	> .55 * *t*

- Subquery quantified test: *List the salespeople who have taken an order that represents more than ten percent of their quota.*

SALESREPS		
EMPL_NUM	*NAME*	*QUOTA*
n	√	*q*

ORDERS	
AMOUNT	*REP*
> .10 * QUOTA	*n*

QBE DATABASE UPDATES

- An update is performed by constructing a example table indicating what operations are to be performed.
- Insert: example insertion using literals and example elements.
- Update: *changeto* operator with expressions and example elements.
- Delete: example deletion using conditions.

EXAMPLE QBE DATABASE UPDATES

- Single-row insert: *Insert a new customer for Mr. Jacobsen.*

CUSTOMERS				
	COMPANY	*CUST_NUM*	*CREDIT_LIMIT*	*CUST_REP*
Insert	'InterCorp'	2126	15000.00	111

- Multi-row insert: *Copy old orders into the OLDORDERS table.*

OLDORDERS			
	ORDER_NUM	*ORDER_DATE*	*AMOUNT*
Insert	*n*	*d*	*a*

ORDERS			
ORDER_NUM	*ORDER_DATE*	*AMOUNT*	
n	*d* < 01-JAN-90	*a*	

- Update: *Transfer all salespeople from the Chicago office to the New York office, and lower their quotas by 10%.*

SALESREPS	
REP_OFFICE	*QUOTA*
12 changeto 11	*q* changeto .90 * *q*

- Delete: *Delete all orders placed before November 15, 1989.*

ORDERS	
	ORDER_DATE
Delete	< 15-NOV-89

IV. ENTITY-RELATIONSHIP MODELLING

DATABASE LIFE CYCLE*

- *Requirements Analysis:* The database requirements are determined by interviewing both the producers and users of data and producing a formal requirements specification. That specification includes the data required for processing; the natural data relationships; and constraints with respect to performance, integrity, and security. It also defines the hardware and software platform for the database implementation.

- *Logical Design:* The *global schema*, which shows all the data and their relationships, is developed using conceptual data modeling techniques such as ER. The data model constructs must ultimately be transformed to normalized (global) relations. The global schema development methodology is the same for either a distributed or centralized database.

* After *T. J. Teorey*.

- *ER modeling:* The data requirements are analyzed and modeled by using an ER diagram that includes, for example, semantics for optional relationships, ternary relationships, and subtypes (categories). Processing requirements are typically specified using natural language expressions or SQL commands along with the frequency of occurrence.

- *View integration:* Usually, when the design is large and more than one person is involved in requirements analysis, multiple views of data and relationship result. To eliminate redundancy and inconsistency from the model, these views must eventually be consolidated into a single global view. View integration requires the use of ER semantic tools such as identification of synonyms, aggregation, and generalization.

- *Transformation of the ER model to SQL relations:* Based on a categorization of ER constructs and a set of mapping rules, each relationship and its associated entities are transformed into a set of candidate relations. Redundant relations are eliminated as part of this process.

- *Normalization of candidate relations:* Functional dependencies (FDs) are derived from the ER diagram. They represent the dependencies among data elements that are keys of entities. Additional FDs and multivalued dependencies (MVDs), which represent the dependencies among key and nonkey attributes within entities, can be derived from the requirements specification. Candidate relations associated with all derived FDs and MVDs are then normalized to the highest degree desired by using standard normalization techniques. Finally, redundancies that occur in normalized candidate relations are analyzed further for possible elimination, with the constraint that data integrity must be preserved.

- *Usage Refinement:* In this step the global schema is refined in limited ways reflecting processing requirements if there are obvious large gains in efficiency to be made. Usage refinement consists of selecting dominant processes on the basis of high frequency, high volume, or explicit priority; defining simple extensions to relations that will improve query performance; evaluating total cost for query, update, and storage; and considering the possible effects of denormalization. The justification for this approach is that, one local site physical design begins, the logical schema is considered to be fixed and is thus a constraint on efficiency.

- *Data Distribution:* Data fragmentation and allocation are also forms of physical design because they must take into account the physical environment, that is, the network configuration. However, this step is separate from local schema and physical design, because design decisions for data distribution are still made independently of the local DBMS. A *fragmentation schema* describes the one-to-many mapping used to partition each global relation into fragments. Fragments are logical portions of global relations, which are physically located at one or several sites of the network. A data *allocation schema* designates where each copy of each fragment is to be stored.

- *Local Schema and Physical Design:* The last step in the design phase is to produce a DBMS-specific physical structure for each of the site databases and to define the *external user schema*. In a heterogeneous system, local schema and physical design are site dependent.

- *Database Implementation, Monitoring, and Modification:* Once the design is completed, the database can be created through implementation of the formal schema by using the data definition language (DDL) of a DBMS. Then, as the database begins operation, monitoring will indicate whether performance requirements are being met. If they are not being satisfied, modifications should be made to improve performance. Other modifications may be necessary when requirements change or end-user expectations increase with good performance.

BASIC ER CONCEPTS*

- *Entity:* An entity is a person, place, or thing, concrete or abstract, which can be uniquely identified, which is relevant to the activity or business we are interested in, and about which we want to store information. An entity set is a type of class or category of entity.

- *Weak Entity:* A weak entity or dependent entity is one which is included only because of its relationship to some other. It cannot stand on its own.

- *Relationship:* A relationship is a specific connection or linkage between two or more entities. Again, we differentiate between a specific occurrence and a type or category. The latter is properly referred to as a relationship set.

* After *C. J. Wertz*.

  - *Binary relationship:* A relationship on two distinct entities.
  - *Ternary relationship:* A relationship on three distinct entities.
  - *Ring, recursive relationship:* A relationship relating an entity to itself.

- *Attribute:* Attributes are the specific data items which describe the entity.
  - *Identifier:* A set of attributes that uniquely identifies an entity or relationship occurrence.
  - *Descriptor:* A nonkey attribute, describing an entity or relationship.
  - *Composite attribute:* A group of simple attributes that together describe a property of an object.

  - *Multi-valued attribute:* An entity attribute that takes on multiple values for a single entity instance.

- *Connectivity:* Connectivity refers to the possibility of a relationship (1-1, 1-M, etc.).

- *Cardinality:* Cardinality refers to the specific number of entities involved in a specific relationship occurrence.

- *Degree:* The number of links or the number of entity types participating in a relationship is referred to as the degree of the relationship.

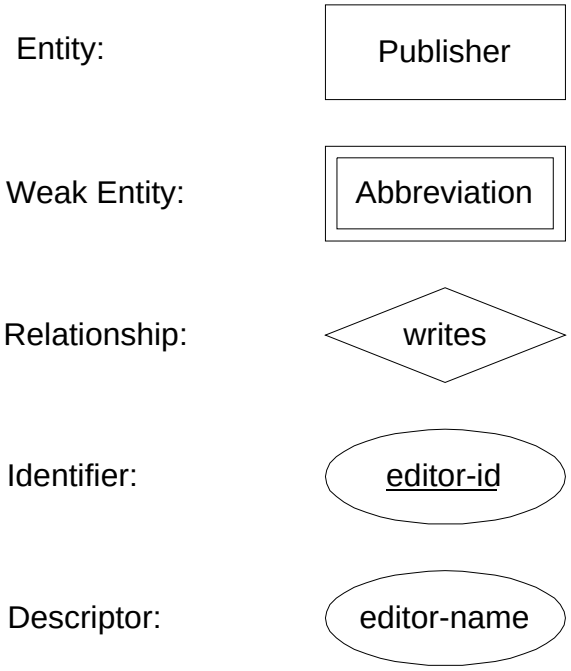
ER MODELING*

- The model and the modeling process are a technique for communication and refining concepts of reality and assuring that the database and system are appropriate to the need.
- Precise definitions and a willingness to discuss and reevaluate are essential.
- The supporting documentation is as important as the diagram.

* After *C. J. Wertz*.

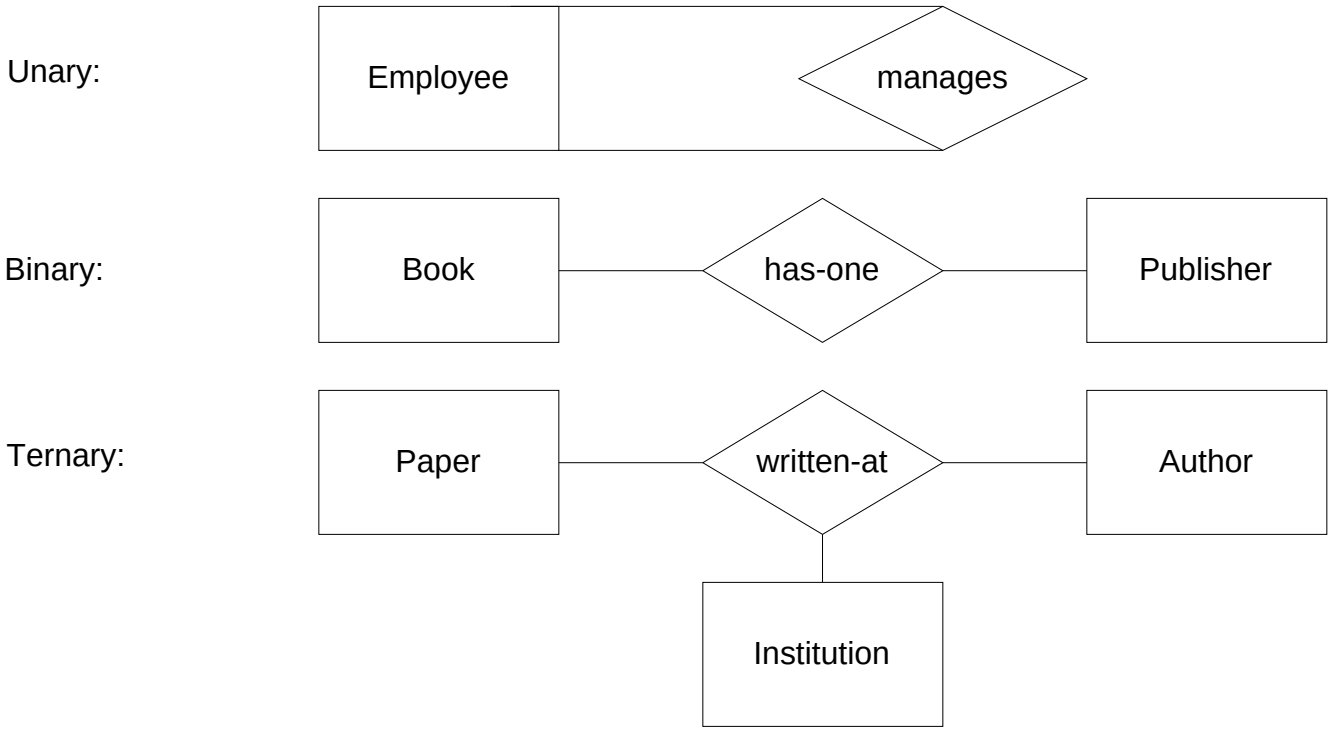
ER DIAGRAMS*

- Basic Objects:

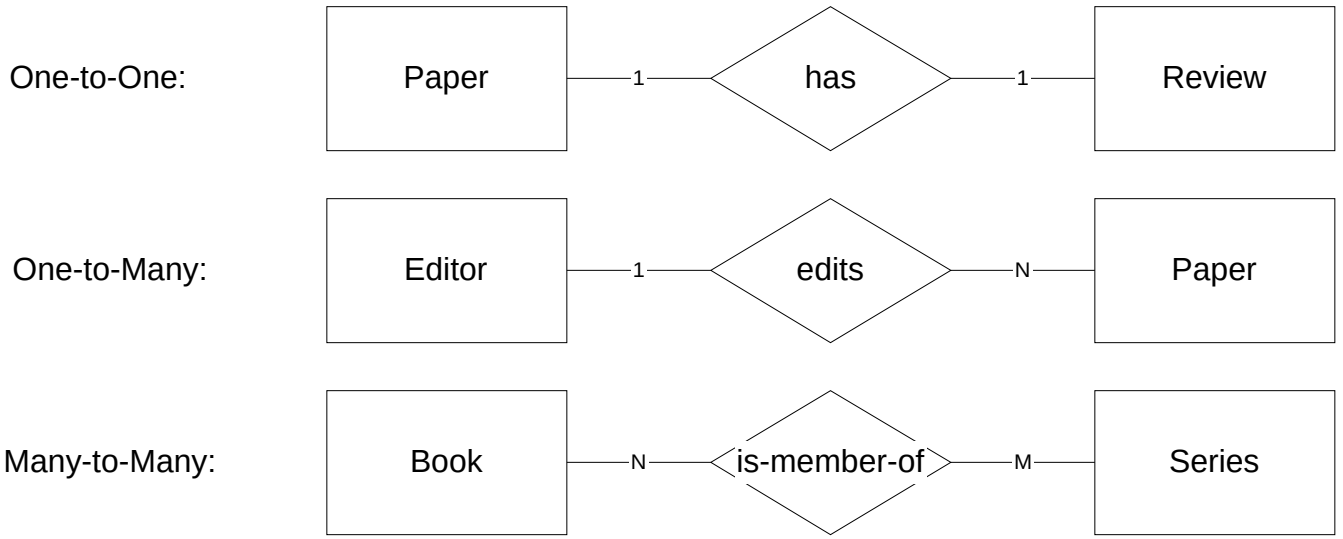


* After *T. J. Teorey*.

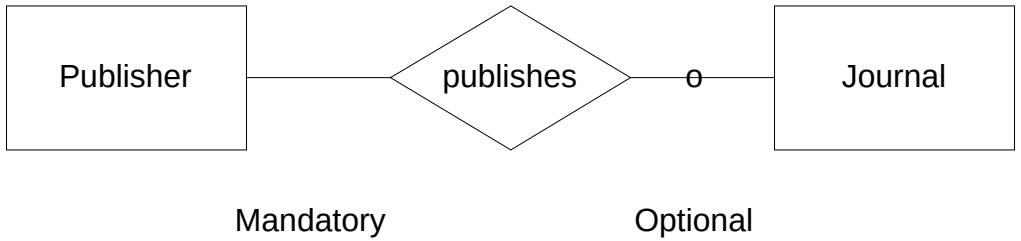
- Degree of Relationships:



- Connectivity:



- Existence:



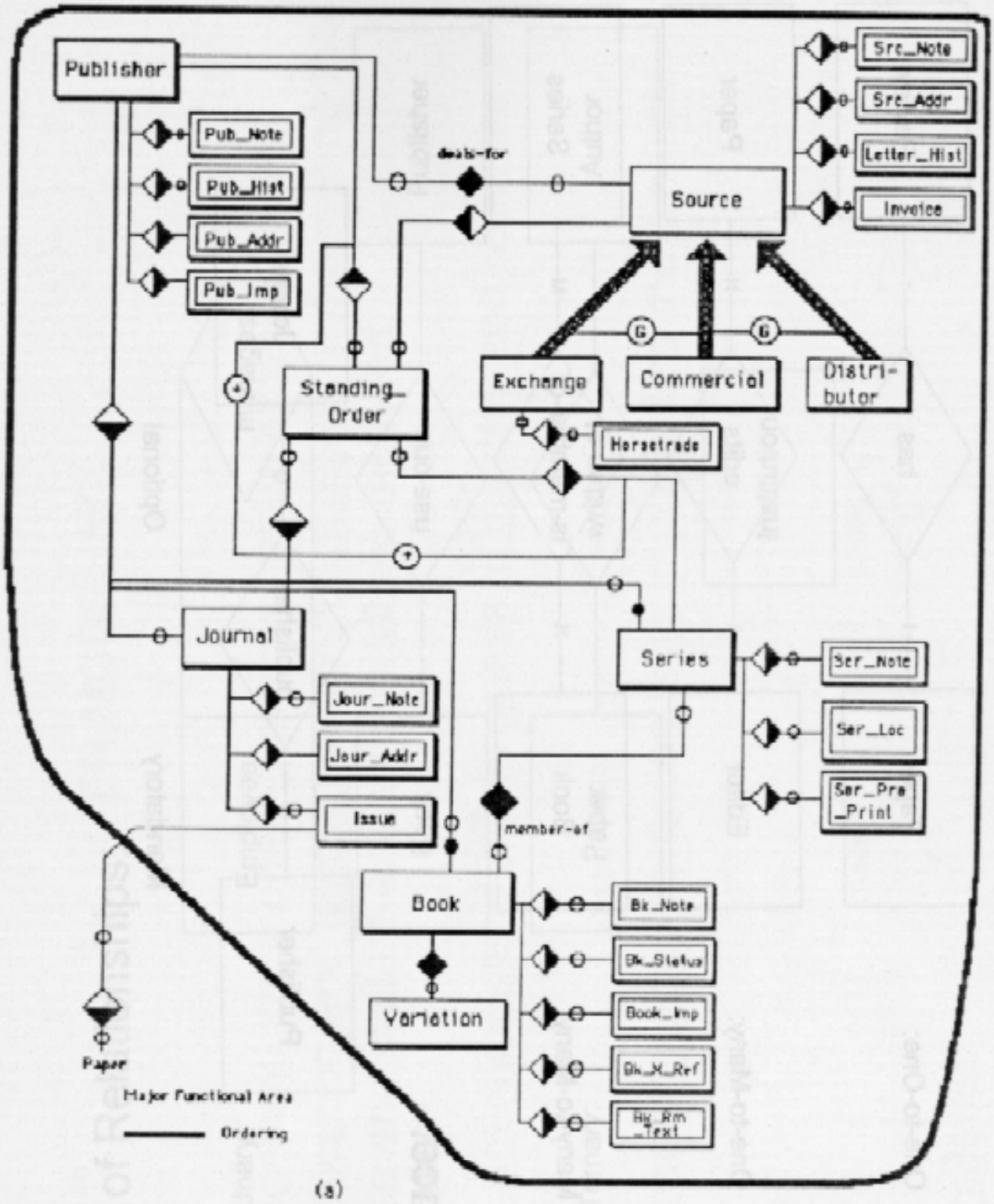


Figure 3.3
Global ER schema and functional areas.

TRANSFORMATION OF ER DIAGRAMS TO RELATIONS*

- Transformations produce three kinds of tables:
  - An entity relation (table) with the same information content as the original entity.
  - An entity relation (table) with the embedded foreign key of the parent entity.
  - A relationship relation (table) with the foreign keys of all the entities in the relationship.

* After *T. J. Teory*.

- Transformation steps:
  - Transform each entity into a table containing the key and nonkey attributes of the entity.
  - Transform every many-to-many binary or unary relationship into a relationship table with the keys of the entities and the attributes of the relationship.
  - Transform every ternary or higher-level n-ary relationship into a relationship table.

RULES FOR NULL VALUES*

- Nulls are allowed only for foreign keys of optional entities in an entity table.
- Nulls are not allowed for foreign keys of mandatory entities in an entity table.
- Nulls are not allowed for any key in a relationship table because only complete rows are meaningful in the table.

* After *T. J. Teory*.

SQL DATA DEFINITION LANGUAGE CONSTRUCTS*

- *Attribute name and data type:* Character, character varying, numeric, decimal integer, small integer, float, double precision, real, date.
- *Not null:* A constraint that specifies that an attribute must have a nonnull value.
- *Unique:* A constraint that specifies that the attribute is a candidate key.
- *Primary key:* A constraint that specifies which candidate key is also the primary key of the table.
- *Foreign key:* The referential integrity constraint specifies that a foreign key in a referencing table column must match an existing primary key in the referenced table.

* After *T. J. Teory*.

- *Deletion of primary key:*
  - *on delete cascade:* The delete operation on the referenced table "cascades" to all matching foreign keys.
  - *on delete set null:* Foreign keys are set to null when they match the primary key of a deleted row in the referenced table.
  - *on delete set default:* Foreign keys are set to a default value when they match the primary key of the deleted row(s) in the reference table.
  - *on delete restrict:* The referenced table rows are deleted only if there are no matching foreign key values in the referencing table.

- *Update of primary key:*
  - *on update cascade:* The update operation on the primary key(s) in the referenced table “cascades” to all matching foreign keys.
  - *on update set null:* Foreign keys are set to null when they match the old primary key value of an updated row in the referenced table.
  - *on update set default:* Foreign keys are set to a default value when they match the primary key of an updated row in the reference table.
  - *on update restrict:* The referenced table rows (primary keys) are updated only if there are no matching foreign key values in the referencing table.

ER TRANSFORMATION EXAMPLES*

- Entity

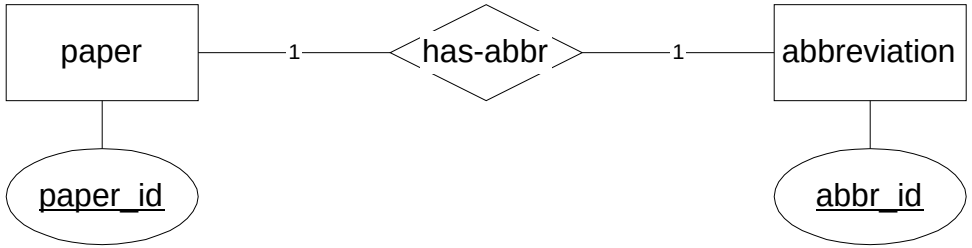

```
create table paper
        (paper_id char(12) not null unique primary key)
```

- Weak entity


```
create table abbreviation
        (abbr_id char(10) not null unique primary key,
        name_id varchar(256) not null unique references name,
        abbr char(10) not null unique)
```

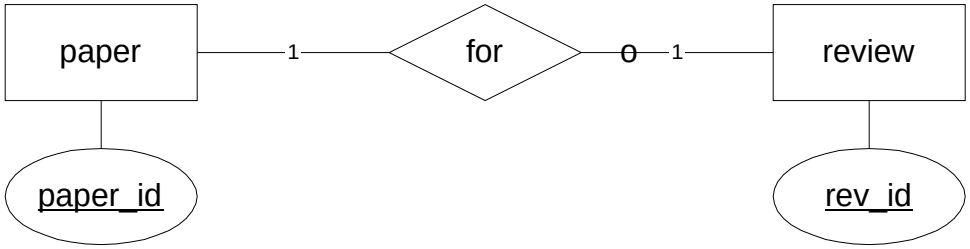
* After *T. J. Teory*.

- Binary relationships
  - One-to-one, both mandatory.



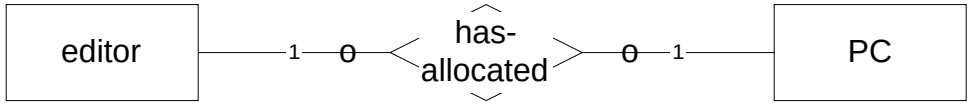

```
create table paper
        (paper_id char(12) not null unique primary key,
        abbr_id char(10) not null unique foreign key references abbreviation)
create table abbreviation
        (abbr_id char(10) not null unique primary key,
        paper_id char(12) not null unique foreign key references paper)
```

– One-to-one, one entity optional.



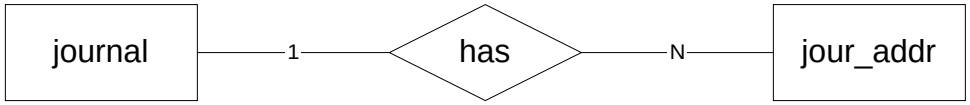

```
create table review
        (rev_id integer not null unique primary key,
        paper_id char(12) not null unique foreign key references paper)
create table paper
        (paper_id char(12) not null unique primary key,
        rev_id integer unique foreign key references review)
```

– One-to-one, both optional.



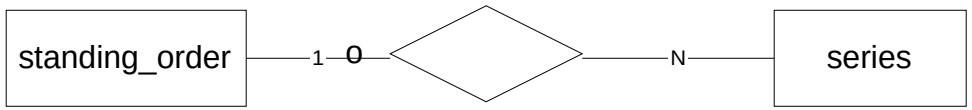
create table editor
(editor_id char(10) not null unique primary key,
pc_id char(15) unique foreign key references pc)
create table pc
(pc_id char(15) not null unique primary key,
editor_id char(10) unique foreign key reference editor)

– One-to-many, both mandatory.



create table journal
(jour_id integer not null unique primary key)
create table jour_addr
(jour_name varchar(256) not null primary key,
jour_id integer not null foreign key references journal on delete cascade,
jour_address varchar(256))

– One-to-many, “one” entity optional.

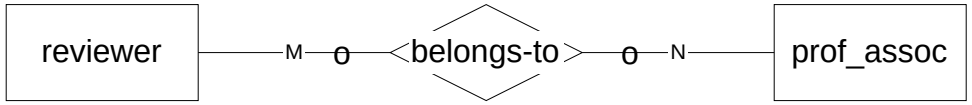


create table standing_order
(st_ord_id integer not null unique primary key)
create table series
(ser_id char(8) not null unique primary key,
st_ord_id integer foreign key references standing_order on delete set null)

– One-to-many, “many” entity optional.

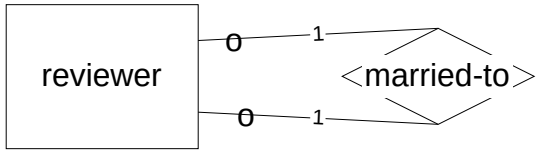

```
create table publisher
        (pub_id char(10) not null unique primary key)
create table journal
        (jour_id integer not null unique primary key,
        pub_id char(10) not null foreign key references publisher)
```

– Many-to-many, both optional.



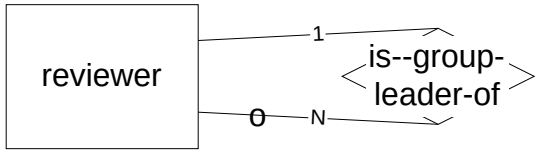

```
create table reviewer
        (rvr_id char(10) not null unique primary key)
create table prof_assoc
        (assoc_name varchar(256) not null unique primary key)
create table belongs_to
        (rvr_id integer not null foreign key references reviewer,
        assoc_name varchar(256) not null foreign key references prof_assoc,
        primary key (rvr_id, assoc_name))
```

- Unary relationships
  - One-to-one, both optional.



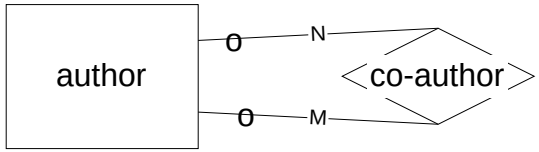

```
create table reviewer
        (rvr_id char(10) not null unique primary key,
        spouse_id integer foreign key references reviewer)
```

  - One-to-many, “many” optional.




```
create table reviewer
        (rvr_id char(10) not null unique primary key,
        leader_id integer not null foreign key references reviewer)
```

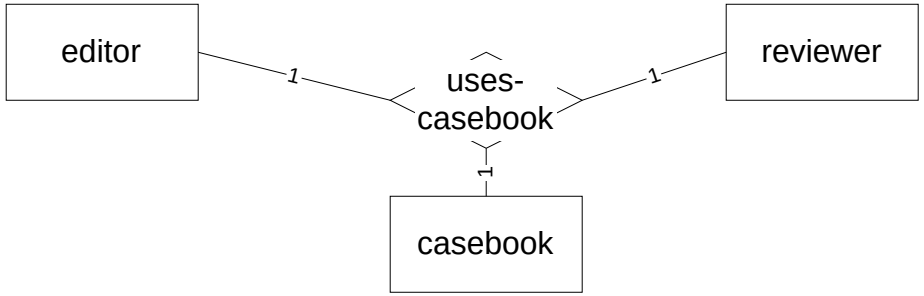
– Many-to-many, both optional.




```
create table author
        (auth_id char(10) not null unique primary key)
create table co_author
        (auth_id char(10) not null foreign key references author,
        co_auth_id char(10) not null foreign key references author,
        primary key (auth_id, co_auth_id))
```

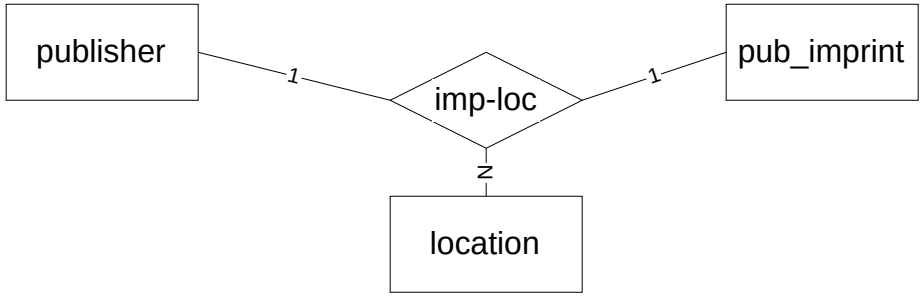
- Ternary relationships

– One-to-one-to-one.



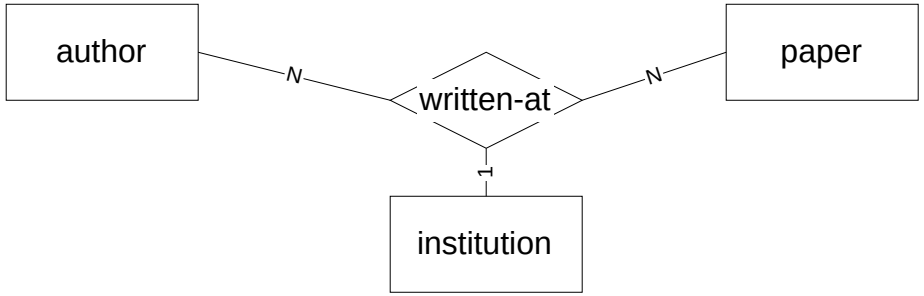

```
create table editor
        (editor_id char(10) not null unique primary key)
create table reviewer
        (rvr_id char(10) not null unique primary key)
create table casebook
        (book_no integer not null unique primary key)
create table uses_casebook
        (editor_id char(10) not null foreign key references editor,
        rvr_id char(10) not null foreign key references reviewer,
        book_no integer not null foreign key references casebook,
        primary key (editor_id, rvr_id),
        unique (editor_id, book_no),
        unique (rvr_id, book_no))
```

– One-to-one-to-many.



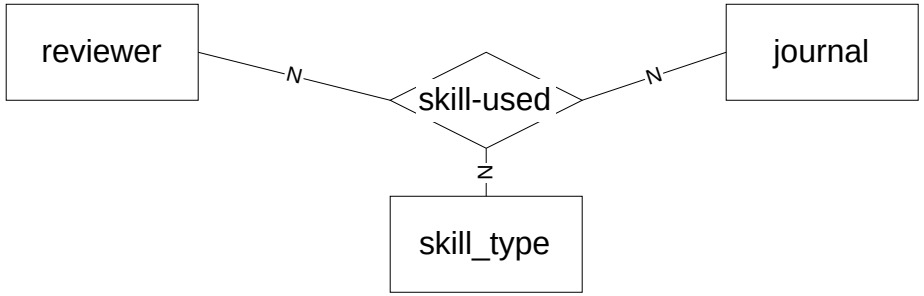

```
create table location
        (loc_addr varchar(256) not null unique primary key)
create table publisher
        (pub_id char(10) not null unique primary key)
create table pub_imprint
        (pub_imp_id char(10) not null unique primary key)
create table imp_loc
        (loc_addr varchar(256) not null foreign key references location,
        pub_id char(10) not null foreign key references publisher,
        pub_imp_id char(10) not null foreign key references pub_imprint,
        primary key (loc_addr, pub_id),
        unique (loc_addr, pub_imp_id))
```

– One-to-many-to-many.




```
create table author
        (auth_id char(10) not null unique primary key)
create table paper
        (paper_id char(12) not null unique primary key)
create table institution
        (inst_name char(20) not null unique primary key)
create table assigned_to
        (auth_id char(10) not null foreign key references author,
        paper_id char(12) not null foreign key references paper,
        inst_name char(20) not null foreign key references institution,
        primary key (auth_id, paper_id))
```

– Many-to-many-to-many.



create table reviewer
(rvr_id char(10) not null unique primary key)
create table skill_type
(skill char(10) not null unique primary key)
create table journal
(jour_id integer not null unique primary key)
create table skill_used
(rvr_id char(10) not null foreign key references reviewer,
skill char(10) not null foreign key references skill_type,
jour_id integer not null foreign key references journal,
primary key (rvr_id, skill, jour_id))

V. FUNCTIONAL DEPENDENCY

DEFINITION*

- A functional dependency (FD) is a relationship between two attributes / data elements / fields and has characteristics similar to those of a function.

- We say that B is functionally dependent on A if and only if there must be a single B value associated with a given A value: $A \rightarrow B$.

- Examples:

  (name_of_stock) $\rightarrow$ (closing_price)
  (student_id, course_id) $\rightarrow$ (grade_received)
  (quantity_of_oranges, price_of_oranges) $\rightarrow$ (price_paid)
  (customer, city) $\rightarrow$ (carrier, shiping_rate)

* After *C. J. Wertz*.

FD CALCULUS*

- *Inclusion Rule:* We are given a table with a specified heading, Head(T). If X and Y are sets of attributes contained in Head(T), and $Y \subseteq X$, then $X \rightarrow Y$.

- *Trivial Dependency:* A trivial dependency is an FD of the form $X \rightarrow Y$ that holds for any table T where $X, Y \subseteq$ Head(T). Given a trivial dependency $X \rightarrow Y$, it must be the case that $Y \subseteq X$.

- *Armstrong's Axioms:*

  – *Reflexive:* $X \rightarrow X$.

  – *Inclusion:* If $Y \subseteq X$, then $X \rightarrow Y$.

* After *P. O'Neil*.

- *Transitivity:* If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.
- *Augmentation:* If $X \rightarrow Y$, then $X \cup Z \rightarrow Y \cup Z$.
- *Union:* If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow Y \cup Z$.
- *Decomposition:* If $X \rightarrow Y \cup Z$, then $X \rightarrow Y$ and $X \rightarrow Z$.
- *Pseudotransitivity:* If $X \rightarrow Y$ and $W \cup Y \rightarrow Z$, then $X \cup W \rightarrow Z$.
- *Accumulation:* If $X \rightarrow Y \cup Z$ and $Z \rightarrow B \cup W$, then $X \rightarrow Y \cup Z \cup B$.

- *Closure of a Set of FDs:* Given a set F of FDs on attributes of a table T, we define the closure of F, symbolized by F^+, to be the set of all FDs implied by F.

- *FD Cover Set:* A set of FDs on a table T is said to cover another set G of FDs on T, if the set G of FDs can be derived by implication rules from the set F, or in other words, if $G \subseteq F^+$.

- *Equivalent FDs:* If F covers G and G covers F, then the two sets of FDs are said to be equivalent, and we write $F \equiv G$.

- *Closure of a Set of Attributes:* Given a set F of FDs on a table T and a set X of attributes contained in T, we define the closure of the set X, denoted by X^+, as the largest subset Y of attributes such that $X \rightarrow Y$ is in F^+.

- Algorithms exists for determining set closures and minimal covers.

- *Decomposition:* Given a table T, a decomposition of T into k tables is a set of tables $\{T_1, T_2, \ldots, T_k\}$ such that $Head(T) = Head(T_1) \cup Head(T_2)$

$\cup . . . \cup$ Head(T_k). Given any specific content of T, the rows of T are projected onto the columns of each T_j during the decomposition.

- *Lossless decomposition:* A decomposition of a table T with a set F of FDs is said to be a lossless decomposition if for any possible future content of T, the FDs F guarantee that the following relationship will hold: $T = T_1 \bowtie T_2 \bowtie \ldots \bowtie T_k$.

USES OF FD CALCULUS

- FD calculus can be used to understand and manipulate the relationships between the various possible candidate keys in a relationship.

- Decompositions can be used to eliminate anomalies in the schema:

  – *update anomaly:* when changing a single attribute value in a row requires that several rows be updated.

  – *delete anomaly:* when deleting an entity causes the loss of information about some other entity.

  – *insert anomaly:* when inserting a new entity relies on information that does not exist.

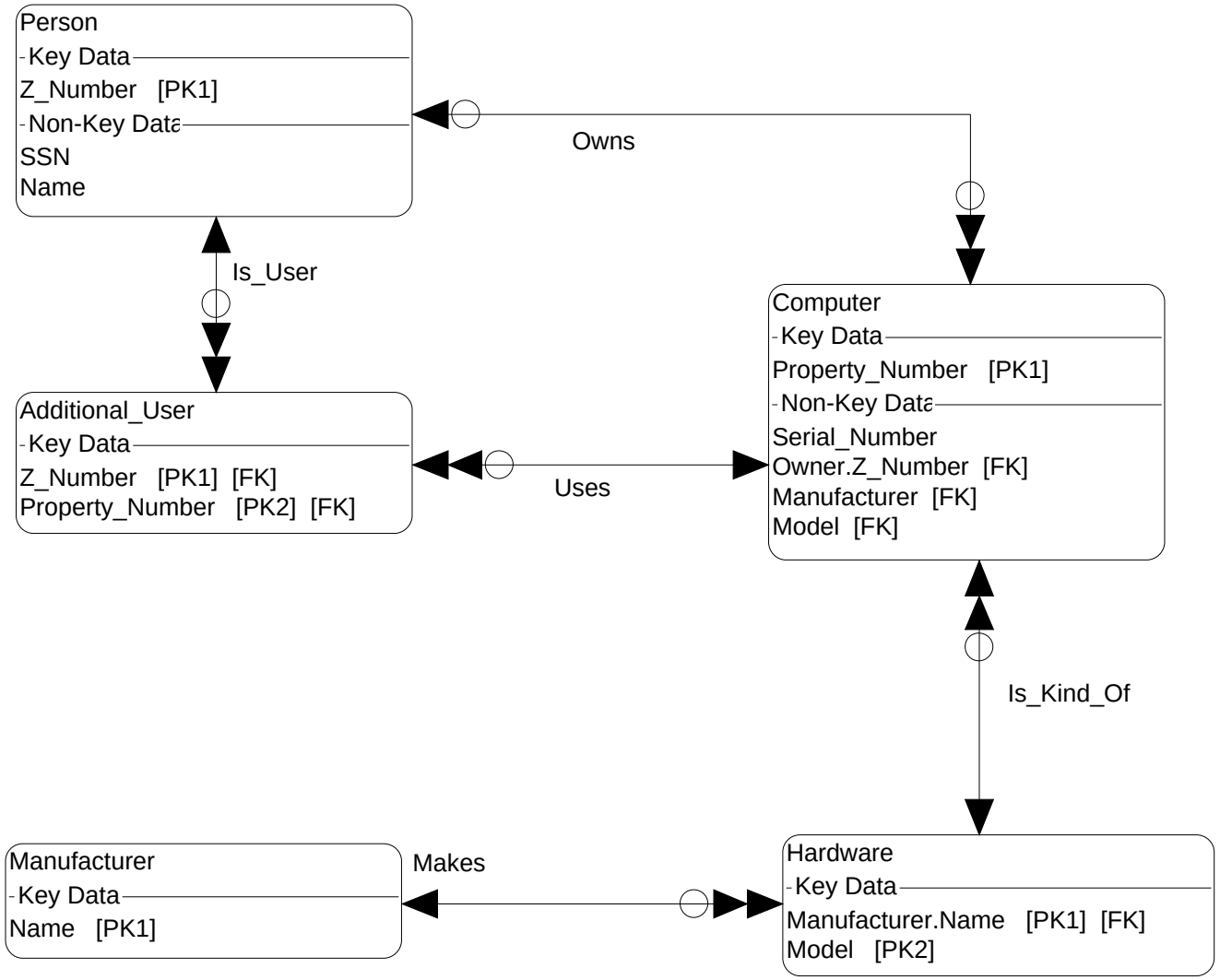
- FD calculus is used in the algorithms for putting a schema into normal form (normalization).

FDs AND ER DIAGRAMS

- Primary FDs can be determined from the ER relationships:

degree	*connectivity*	*FDs*
unary / binary	one-to-one	$A \rightarrow B$ $B \rightarrow A$
	one-to-many	$B \rightarrow A$
	many-to-many	$A \cup B \rightarrow \varnothing$
ternary	one-to-one-to-one	$A \cup B \rightarrow C$ $B \cup C \rightarrow A$ $C \cup A \rightarrow B$
	one-to-one-to-many	$A \cup C \rightarrow B$ $B \cup C \rightarrow A$
	one-to-many-to-many	$B \cup C \rightarrow A$
	many-to-many-to-many	$A \cup B \cup C \rightarrow \varnothing$

- Additional FDs can be determined by examining candidate relations.


Person
Key Data
Z_Number [PK1]
Non-Key Data
SSN
Name
Owns
Is_User
Computer
Key Data
Property_Number [PK1]
Non-Key Data
Serial_Number
Owner.Z_Number [FK]
Manufacturer [FK]
Model [FK]
Additional_User
Key Data
Z_Number [PK1] [FK]
Property_Number [PK2] [FK]
Uses
Is_Kind_Of
Manufacturer
Key Data
Name [PK1]
Makes
Hardware
Key Data
Manufacturer.Name [PK1] [FK]
Model [PK2]

VI. NORMALIZATION

WHAT IS NORMALIZATION?

- Normalization is used to determine which attributes to put into which relations.
- Normalization's goal is to create relations that obey the conditions for normal forms (1NF, 2NF, 3NF, BCNF, 4NF, 5NF).
- Normalization is closely related to ER modeling.
- Normalization makes certain anomalies impossible: *Successive decomposition removes anomalies.*

FIRST NORMAL FORM (1NF)

- Definitions:
  - All attributes are atomic.*
  - A relation is in 1NF if and only if all underlying domains contain only atomic values.†
  - The rule for 1NF is eliminate repeating groups.‡
  - 1NF means that every column is a scalar (or atomic) value.§

* After *A. L. Pope*.
† After *T. J. Teory*.
‡ After *C. J. Wertz*.
§ After *D. W. Hubbard* and *J. Celko*.

- Example:

  FDs:
  Paper_ID $\rightarrow$ Title
  Paper_ID $\rightarrow\rightarrow$ {Author}

  Unnormalized:
  Paper(Paper_ID, Title, Author_1, Author_2, Author_3)

  1NF:
  Paper(Paper_ID, Title)
  Authorship(Paper_ID, Author)

SECOND NORMAL FORM (2NF)

- Definitions:
  - Functional dependency is not commutative between attributes.*
  - A relation is in 2NF if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key.†
  - The rule for 2NF is all nonkey attributes of a relation must be functionally dependent on the entire key.‡
  - A table is in 2NF if it has no “partial key” dependencies; that is, if X and Y are columns, and X is a key, then for any Z that is a proper subset of X, it cannot be the case that $Z \rightarrow Y$.§

* After *A. L. Pope*.
† After *T. J. Teory*.
‡ After *C. J. Wertz*.
§ After *D. W. Hubbard* and *J. Celko*.

- Example:

FDs:
Paper_ID $\rightarrow\!\!\rightarrow$ {Author}
Author $\rightarrow$ Address
Paper_ID $\rightarrow\!\!\rightarrow$ {Address}

1NF:
Authorship(Paper_ID, Author, Author_Address)

2NF:
Authorship(Paper_ID, Author)
Author_Info(Author, Address)

THIRD NORMAL FORM (3NF)

- Definitions:
  - One attribute determines another, which it does not contain, and one is key.*
  - A relation is in 3NF if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.†
  - The rule for 3NF is transitive dependencies are not allowed; each nonkey attribute of a relation must be functionallly dependent on the key and nothing else.‡
  - A table is in 3NF if for each X and Y in the table where X $\rightarrow$ Y, X is a key or Y is a candidate key.§

* After *A. L. Pope*.
† After *T. J. Teory*.
‡ After *C. J. Wertz*.
§ After *D. W. Hubbard* and *J. Celko*.

- Example:

FDs:
Paper_ID → Title
Paper_ID → Institution
Institution → Address
Paper_ID → Address

2NF:
Paper(Paper_ID, Title, Institution, Institution_Address)

3NF:
Paper(Paper_ID, Title, Institution)
Institution_Info(Institution, Address)

BOYCE-CODD NORMAL FORM (BCNF)

- Definitions:
  - One attribute determines another and is either the key or superkey.*
  - A relation is in BCNF if and only if for every nontrivial FD $X \rightarrow A$, X is a superkey.†
  - The rule for BCNF is that every determinant must be a candidate key.‡
  - BCNF requires the removal of transitive dependencies regardless of whether a resultant attribute is part of a candidate key.§

* After *A. L. Pope*.
† After *T. J. Teory*.
‡ After *C. J. Wertz*.
§ After *D. W. Hubbard* and *J. Celko*.

- Example:

FDs:
Reviewer_Name $\cup$ Team_ID $\rightarrow$ Editor_Name
Editor_Name $\rightarrow$ Team_ID

3NF:
Reviewer_Assignment(Reviewer_Name, Team_ID, Editor_Name)
Editor_Assignment(Editor_Name, Team_ID)

BCNF:
Reviewer_Assignment(Reviewer_Name, Editor_Name)
Editor_Assignment(Editor_Name, Team_ID)

FOURTH NORMAL FORM (4NF)

- Definitions:
  - One attribute multi-determines others and is either key or superkey.*
  - A relation is in 4NF if and only if it is in BCNF and, whenever these exists a multivalued dependency (MVD) in R (say $X \rightarrow Y$), at least one of the following holds: (a) the MVD is trivial, or (b) X is a superkey of R.†
  - The rule for 4NF requires us to eliminate anomalies by eliminating multivalued dependencies.‡

* After *A. L. Pope*.
† After *T. J. Teory*.
‡ After *C. J. Wertz*.

- Example:

FDs:
Paper_ID $\rightarrow\!\!\rightarrow$ {Skill}
Paper_ID $\rightarrow\!\!\rightarrow$ {Reviewer}
Skill $\rightarrow\!\!\rightarrow$ {Paper_ID)
Skill $\rightarrow\!\!\rightarrow$ {Reviewer}
Reviewer $\rightarrow\!\!\rightarrow$ {Paper_ID}
Reviewer $\rightarrow\!\!\rightarrow$ {Skill}

BCNF:
Skill_Required(Paper_ID, Skill, Reviewer)

4NF:
Skill_Required_for_Paper(Skill_ID, Paper_ID)
Skill_Required_for_Reviewer(Skill_ID, Reviewer)

FIFTH NORMAL FORM (5NF)

- Definitions:
  - A relation is in 5NF if it cannot have a lossless decomposition/join by the projection operator into any number of smaller relations.[†]
  - The rule for 5NF is do not create sets of relations which are susceptible to lossy joins; be sure that the relations do represent the facts to be recorded.[‡]

[†] After *T. J. Teory*.
[‡] After *C. J. Wertz*.

- Example:

FDs:
Paper_ID $\rightarrow\!\!\rightarrow$ {Skill $\cup$ Reviewer}
Skill $\rightarrow\!\!\rightarrow$ {Paper_ID $\cup$ Reviewer)
Reviewer $\rightarrow\!\!\rightarrow$ {Paper_ID $\cup$ Skill}

4NF:
Skill_Required_for_Paper(Skill_ID, Paper_ID)
Skill_Required_for_Reviewer(Skill_ID, Reviewer)

5NF:
Skill_Required(Paper_ID, Skill, Reviewer)

3NF SYNTHESIS ALGORITHM*

- Eliminate extraneous attributes in the determinants of FDs .
- Search for a nonredundant cover.
- Partition the cover into groups so that all FDs with the same left side are in one group
- Merge equivalent keys.
- Search for a nonredundant cover to eliminate transitive Fds.
- Construction of relation schemes and FDs.

* After *T. J. Teory*.

VII. DISTRIBUTED DATABASES

TYPES OF DATABASE ARCHITECTURES

- *Single User:* one user, one CPU, one data repository.
- *Multiple User:* several users, one CPU, one data repository.
- *Client/Server:* several users, several CPUs, one data repository.
- *Distributed:* several users, several CPUs, several data repositories.

DATE'S RULES FOR DISTRIBUTED DATABASE SYSTEMS*

- *Fundamental Principle:* To the user, a distributed system should look exactly like a non-distributed system.
- *Local Autonomy:* The sites in a distributed system should be autonomous.
- *No Reliance on a Central Site:* There must not be any reliance on a central or master site.
- *Continuous Operation:* There should never be any need for a planned system shutdown.
- *Location Independence:* Users should not have to know where data is physically stored.

* After *C. J. Date*.

- *Fragmentation Independence:* Data can be stored at the location where it is most frequently used.
- *Replication Independence:* A relation . . . can be represented by many distinct stored copies.
- *Distributed Query Processing:* Optimization . . . is crucial.
- *Distributed Transaction Management:* There are two major aspects to transaction management: recovery control and concurrency control.
- *Hardware Independence:* It is desirable to be able to run the same DBMS on different hardware systems.
- *Operating System Independence:* It is desirable . . . also to be able to run the same DBMS on different operating systems.

- *Network Independence:* It is . . . desirable to support a variety of disparate communication networks also.

- *DBMS Independence:* All that is needed is that the DBMSs at different sites all support the same interface.

DISTRIBUTED DATABASE PROS AND CONS*

- Advantages:
  - local autonomy
  - improved performance
  - improved reliability
  - improved availability
  - economics
  - expandability
  - shareability
  - reduced data transmission
  - tuning
  - user control

* After *C. J. Wertz*.

- Disadvantages:
  - lack of experience
  - complexity
  - cost
  - distribution of control
  - security
  - difficulty of change
  - tuning

VIII. PROGRAMMATIC DATABASE MANIPULATION

PROGRAMMATIC DATABASE MANIPULATION

- It is often desirable to manipulate a database using a programming language rather than just interactively.
- There are several approaches to programmatic manipulation:
  - Embed the database language in the programming language.
  - Supply a set of functions callable from the programming language
  - Provide an object-oriented framework for database manipulation.
  - Pass messages between the application program and the database server.

IX. OBJECT-ORIENTED DATABASES

ODB DEFINITION

- Database capabilities:

persistence	performance	versioning
concurrency	security	recovery
transactions	integrity	querying

- Object orientation

abstract data typing
inheritance
object identity

ODB FEATURES*

- *Base types:* A base type is one that is primitive to the system.
- *Extensibility of Base Types:* New base types can be added to the system.
- *Composite Types:* A composite type has component data values (like a “struct” in C or “record” in Pascal).
- *Complex Objects:* Complex objects are collections of objects or references to objects (“set,” “array,” “reference,” etc.).
- *Encapsulation:* Data cannot be manipulated directly.
- *Functions:* The object can manipulate its own data.

* After *P. O’Neil*.

- *Type Hierarchies:* A type can be defined as a subtype (extension or restriction) of another type.

APPROACHES TO ODB*

- Novel database data model/data language approach.
- Extending an existing database language with object-oriented capabilities.
- Extending an existing object-oriented programming language with database capabilities.
- Providing extendable object-oriented database management system libraries.
- Embedding object-oriented database language constructs in a host (conventional) language.

* After *S. Khoshafian*.

- Application-specific products with an underlying object-oriented database management system.